iLoc: A Framework for Incremental Location-State Acquisition and Prediction based on Mobile Sensors

Yiming Ma, Rich Hankins, and David Racz
Nokia Research Center
Palo Alto, California, USA
yiming.ma@nokia.com, rich.hankins@nokia.com, david.racz@nokia.com

ABSTRACT

Much research focuses on predicting a person's geo-spatial traversal patterns using a history of recorded geo-coordinates. In this paper, we focus on the problem of predicting *location-state* transitions. Location-states for a user refer to a set of anchoring points/regions in space, and the prediction task produces a sequence of predicted location states for a given query time window. If this problem can be solved accurately and efficiently, it may lead to new location based services (LBS) that can smartly recommend information to a user based on his current and future location states. The proposed *iLoc* (***I***ncremental ***Loc***ation-State Acquisition and Prediction) framework solves the prediction problem by utilizing the sensor information provided by a user's mobile device. It incrementally learns the location states by constantly monitoring the signal environment of the mobile device. Further, the framework tightly integrates the learning and prediction modules, allowing *iLoc* to update location-states continuously and predict future location-states at the same time. Our extensive experiments show that the quality of the location-states learned by *iLoc* are better than the state-of-the-art. We also show that when other learners failed to produce reasonable predictions, *iLoc* provides good forecasts. As for the efficiency, *iLoc* processes the data in a single pass, which fits well to many data stream processing models.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data ming Spatial databases and GIS*

General Terms

Algorithms, Experimentation, Performance, Reliability

1. INTRODUCTION

The increasingly sophisticated set of sensors built into mobile devices have recently attracted tremendous amount of R&D attention. The capabilities of a modern mobile phone far surpass those necessary for making simple voice calls. For example, today's smartphone includes WLAN networking, a GPS receiver, an accelerometer, a high resolution camera, and many other sensors. Commercial and research applications are already exploiting this rich set of sensing information from mobile devices, and common across many of these applications is the use of location information. Given the importance of location information, much effort has been put into accurate and efficient location sensing.

Interestingly, current approaches typically separate the location information from the state information. For example, many social network services directly utilize geo-coordinates gathered from mobile devices, but state information such as "at home" or "at work" is either not collected, or not fully connected to the location information. Mining location-state information may enable more semantically enriched information retrieval (e.g., retrieve today's call entries when I was at office). And utilizing the state information for predicting users' intentions may empower a new set of smart content serving applications. For example, if a service wants to push lunch coupons for a particular restaurant to a set of users based only on their current geo-coordinate, it cannot differentiate if the users intend to have lunch or are simply passing by on their way to work. By accurately predicting a user is leaving for lunch, the relevancy of the lunch coupon increases greatly.

In this paper, we propose a novel framework called *iLoc* (***I***ncremental ***Loc***ation-State Acquisition and Prediction). This framework provides two components critical to a new generation of location-based services, namely the efficient acquisition of location-state information and a predictive model to forecast future location-state transitions. More specifically, the primary contributions of this paper include:

- An unsupervised incremental location-state learning and predication framework (Section 4).
- A set of effective, yet practical, implementations of various learning and prediction components (Section 5, 6).
- An extensive empirical evaluation to validate different learning and prediction components, based on over two months of real sensing data from actual users (Section 7).

2. RELATED WORK

In the following discussion, we use an example dataset to better illustrate related approaches for location-state determination and prediction. Table 1, shows data collected using a mobile device's WiFi scanner sampling the environ-

tid	*Time*	*Observations*
1	8AM	{homeWiFi, homeWiFi2}
2	8:02 AM	{homeWiFi, homeWiFi2, homeWiFi3}
...	...	...
15	8:28 AM	{PublicWiFi}
...	...	...
30	8:58 AM	{workWiFi, workWiFi2, workWiFi3}
31	9 AM	{workWiFi}

Table 1: Example WiFi examples

ment every 2 minutes. Each scan generates a set of observations, also called WiFi fingerprints. The first 2 records (*tid1*, *tid2*) are observed when the user is at home, and the last 2 records (*tid30*, *tid31*) are at the office. The middle record (*tid15*) represents a WiFi access point observed during the transition from home to work.

The problem of mining location states from signal inputs can be viewed as an unsupervised learning task, where many clustering algorithms can be applied. However, we notice that the data is highly unstructured; at each sampling point multiple signal sources can be observed, and the number of potential signal sources can be in the thousands. Therefore, the location state discovery problem is very similar to the topic mining problem in the text mining domain. The leading method for this task is Latent Dirichlet Allocation (LDA) [3,5,13]. LDA is a generative model that uses a mixture of multinomials with Dirichlet priors. In our example, an observed signal source in a record is generated from a mixture of latent state variables, and each has a multinomial distribution. Although LDA has shown very good performance in text mining domains, there are three problems when applied to our use case. First, LDA does not consider time dependencies. Each record/document is assumed to be independent or interchangeable. Therefore, an LDA model cannot be directly used in predicting future location states. Second, LDA does not have the semantic concept of stationary states, which represent the places where a user spends most of his time. For instance, the first two and last two records in Table 1 can be considered from two stationary states, but not the middle one. However, in a LDA model, the WiFi source in the third record will appear in both of the stationary states, which leads to poor clustering performance. Third, it is difficult to incrementally update the LDA model, especially when deciding the number of clusters and estimating the parameters. Although other non-model based clustering algorithms, such as K-means or hierarchical agglomerative clustering (HAC), can also be applied, they have been shown less accurate than model-based methods [8]. Furthermore, these non-model based methods suffer from similar problems as LDA.

Since our data records have time dependencies, sequence mining techniques can be applied. The most popular modeling method is hidden Markov model (HMM) [7,10]. HMM is also a generative model, such that each observation is generated by a latent state variable. The latent state at a time point also has a dependency on its previous latent state. The parameters in the HMM can be estimated by using Baum-Welch algorithm, which is a generalized expectation-maximization (GEM) algorithm. Although modeling state transition is part of HMM, the states do not physically correspond to the actual location states. This leads to poor performance in predicting future movements. For HMM, the typical predication task is to predict the best set of latent state sequences for a given observation sequence, and the

Symbols	*Definitions*
z_i	a stationary state in $\mathbb{Z}$
s_j	a transitional state in $\mathbb{S}$ consists of a pair of stationary states.
t_i	a time instance or a time tick interval (e.g., 2 minutes)
Λ	a sensor input stream
υ_t	sensor input at time instance t
$\mathcal{C}$	movement classifier
$\mathcal{M}$	a similarity function
$\mathcal{F}$	a temporal feature extraction function
α	an input signal source (e.g., a WiFi SSID)
β	a temporal feature-value pair
γ	a time duration function

Table 2: Definitions of main symbols

Viterbi algorithm is designed for this task. This algorithm finds the most likely sequence of hidden states given a sequence of observed events. This is not sufficient for our problem, as the observation sequence is not available at the point of prediction. For example, only given the first 2 records in our sample data set, we would like to predict the transition between the two stationary location states. Without providing future observations, HMM is very unlikely to predict the transition. However, we adopt the Viterbi algorithm in our prediction process. By further incorporating time and movement information, we are able to accurately predict the transition behaviors.

Many high-end mobile devices are equipped with GPS receivers. Unfortunately, power issues typically restrict the use of the phone's GPS as a reliable sensor. For instance, continuous usage of the built-in GPS receiver will drain the phone's battery in less than a few hours. In addition, GPS receivers may not work in indoor situations. Recent work has shown that one can use WiFi or GSM signal fingerprints for localization [4,9,12], which may overcome some of the energy problems with GPS. These methods have been shown to accurately estimate actual geo-coordinates. While we do not depend on accurate geo-coordinates, when available, our framework can use the information to improve the clustering quality.

In [6], the authors propose an algorithm called Beacon-Print that uses WiFi and GSM radio fingerprints to automatically learn the places a user visits, and detects when the user returns to those places. This is similar to the algorithm used in this paper for movement detection. However, unlike our approach, Beacon-Print stops short in using the movement information to incrementally cluster the signal fingerprints into stationary states, and does not combine time and stationary states to predict the future movements of a user.

3. DEFINITIONS AND PROBLEM STATEMENT

3.1 Definitions

In this section, we provide the definitions to various terms used in this paper. Table 2 summarizes the major symbols.

Location-state: It corresponds to two possible states – a stationary state z or a transitional state s. Stationary states $\mathbb{Z}$ represent a set of repeatable geographical region that a user frequently "stays" in. Transitional states $\mathbb{S}$ represent transitional behaviors among the stationary states.

Time Tick (t_i)***:*** We partition the time dimension into fixed intervals (e.g., 2 minutes). A time tick is the smallest unit of time. Therefore, we use time instance and time tick interchangeably in the reminder of the paper.

Sensor stream: A sensor stream (Λ) represents a continuous set of measures from a sensor. At a time instance (t), a sensor measure can be viewed as a data point (v) in a feature space (F). Based on different sampling schemes, a sensor stream can also be represented as a set of data points with time stamps (i.e., $\{v_1 \ldots v_i \ldots\}$). In this paper, we focus on two types of sensor streams: Λ_{gsm} and Λ_{WiFi}, representing GSM- and WiFi-related sensor streams respectively.

3.2 Problem Statement

This paper focuses on the following two problems:

- ***Learning problem:*** Given a sensor stream (Λ), automatically learn and refine the stationary states $\mathbb{Z}$ and transitional states $\mathbb{S}$.
- ***Predicting problem:*** At a time tick t_i and a prediction time window $w = \{t_i, \ldots t_{i+k}\}$, given a predictive model that is built by using the observations prior to t_i, predict a sequence of transitional states $\{s_i \ldots s_{i+k}\}$.

Obviously, these two problems are related. If we can solve the first problem accurately and efficiently, we can use the resulting models for the second problem. Also, we note that our prediction problem focuses on a future time window with unknown observations. Although our solution can be applied to predicting location states for historical observations, forecasting future location states is a more challenging and practically useful problem to solve.

4. ARCHITECTURE AND OVERVIEW

Figure 1 shows the overall framework of *iLoc*. As shown in the figure, the three major components of the framework are: annotation, learning, and prediction. While this paper focuses on the learning and prediction components of the framework, we first briefly describe all three components.

The learning component consists of two sub-components: a movement detector and an incremental state learner. The movement detector determines movement (i.e., stationary, moving) and attaches movement information to the input data stream. Detection happens by comparing a set of current data points with previous observations, and then determining if significant changes have occurred. The incremental state learner maintains the stationary and transitional models by utilizing the outputs from movement detector. Section 5 provides further detail on these two sub-components.

The prediction component uses the output from the movement detector, along with the the location-state models, to predict a user's future location states. The problem formalization and detailed implementations are presented in Section 6.

The annotation component connects the user's tagged states to the learned states in the learning component's stationary model, providing additional semantic information to the generated output states (e.g., "at home" or "at work"). While not the focus of this paper, we briefly describe how the tagging information can be effectively obtained. Tags can be collected passively or actively, or even a combination of the two methods. In passive collection, a server can listen to the content pushed from a mobile device (e.g., micro-blogging), and based on this content, the location tagging information can be derived. In active collection, the user is prompted

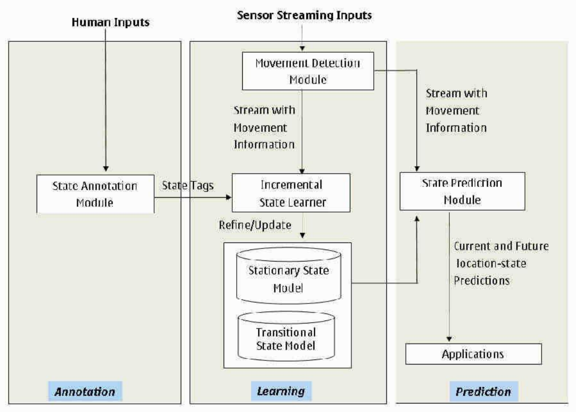


Figure 1: Overall architecture

to provide the needed information, such as prompting the user to confirm different location tags when the system believes the user is in a stationary state. As we will show in Section 7, our learner can quickly and accurately derive the frequent location states. The system needs to ask only a very few questions to the user, which makes the active approach feasible and less annoying to the user.

5. LEARNING COMPONENTS

In this section, we discuss the implementation of the iLoc framework's learning component.

5.1 Movement Detector

A movement detector is essentially a classifier $\mathcal{C}$ that has two output classes (i.e., *move*, *stay*). Given a sensor input v_t at a time instance t, it attaches a movement decision (i.e., $\mathcal{C}(v_t)$) to v_t.

In general, building such a classifier requires a set of labeled training examples that reflect a user's movement behavior. One way to obtain such a training set is to regularly prompt the user to classify his movements, and then attach sensor inputs to these movement tags. A classifier can then be trained to classify the movements. However, in practice it is difficult to optimize the rate at which a user is prompted. If prompted frequently, the user may become annoyed and stop providing movement information. Prompting infrequently may not provide enough information to cover a user's day to day life.

For the purpose of discovering location states, we found that signal fingerprints work well. Instead of using them for the purpose of localization [4, 9, 12], we use them to detect movements. The basic idea is that if a user is stationary, the corresponding signal fingerprints should remain relatively similar to each other. To further develop this idea, we first partition the time into a set of fixed length time intervals w (e.g., 10 minutes). We maintain the two most recent time windows (a sliding window) for an input stream Λ. If a similarity function $\mathcal{M} \in [0, 1]$ is defined on these two sets, the classifier can then be built by imposing a threshold value τ on the outcome of the similarity function $\mathcal{M}$. If the similarity value is bigger than the threshold, we classify the current input v_t as *not moving*, and *moving* otherwise. The overall classifier $\mathcal{C}(v_t)$ can then be defined as:

$$\mathcal{C}(v_t) = [\mathcal{M}(\{v_t \cdots v_{t-w}\}, \{v_{t-w-1} \cdots v_{t-2w}\})] \geq \tau$$

Since we focus on the location-states, the sensor inputs should correspond to some fixed location regions. GSM and

WiFi data streams are good candidates for this purpose. For GSM and WiFi, we can define the similarity function using cosine similarity, commonly used by many information retrieval systems [11]. For the two time windows, we group the observations into 2 vectors $(v1, v2)$. The cosine similarity can then be defined as: $\mathcal{M} = \frac{v1 \cdot v2}{|v1| \times |v2|}$.

In Section 7, we demonstrate that this simple approach successfully captures a large amount of a user's daily movements, and hence provides a reliable input to the subsequent mining components.

5.2 Stationary State Learner

The goal of the stationary state learner is to efficiently and accurately determine the stationary states, which can then be used for the purpose of modeling state transitional behavior. Figure 2 outlines the algorithm. (Note that, in this section we focus on discovering and maintaining the stationary states $\mathbb{Z} = \{z_1 \dots z_k\}$, and postpone discussion of the transitional behavior until the following section.)

The algorithm begins by initializing the stationary states $\mathbb{Z} = \{\}$. At a time instance i, the movement detector decides if the stream input v_i belongs to a stationary state. If so, the learner checks if the input belongs to an existing state or is a new state. If it is found in state z_i, it updates the cluster representation; otherwise, a new cluster z_{new} is added to the stationary states.

The key component to this algorithm is the classification process at line 6. Essentially, it computes the posterior distribution of $Pr(z|v_i)$. Each input can be decomposed into a set of feature-value pairs, also called attribute-value pairs. For example, a WiFi signal input v can have a set of k WiFi access points or SSIDs $\{\alpha_1 \dots \alpha_k\}$, where each of the SSID can be treated as a binary feature, and the input represents a set of visible SSIDs (i.e., $v_i = \{\alpha_1 \dots \alpha_k\}$). The posterior distribution can be estimated as:

$$Pr(z|v_i) = \frac{Pr(z) \times \prod_{i=1}^{i=k} Pr(\alpha_i|z)}{\sum_{j=1}^{|\mathbb{Z}|} Pr(z_j) \times \prod_{i=1}^{i=k} Pr(\alpha_i|z_j)}$$

The conditional independent assumption $Pr(\alpha_1 \dots \alpha_k|z) = \prod_{i=1}^{i=k} Pr(a_i|z)$ is similar to the Naive Bayes (NB). It is a reasonable assumption since sensor features (e.g., WiFi SSIDs), very often, are independently installed. Given the posterior distribution of $Pr(z|v_i)$, we can take the best class/state for the current input. We can then use an accuracy threshold τ to establish the quality of the classification. If the best state has the posterior probability less than the threshold, we do not assign the state label, and a new cluster is created (line 11). The model parameters $Pr(\alpha_i|z)$ and $Pr(z)$ can be easily updated at line 8.

The above approach works when each $Pr(\alpha_i|z) \neq 0$, otherwise the likelihood probability for the input vector $Pr(v_i|z)$ becomes 0. We handle this case in two different ways. In the first scenario, if a large portion (e.g., 70%) of the terms in a input vector are missing from a cluster z, the likelihood probability for that cluster becomes 0. Furthermore, if likelihood probabilities for all the clusters are 0, a new cluster will be generated. In the second scenario, only a small portion of the terms in the input vector are missing given a cluster z, each of the missing term α_i is assigned with a small likelihood probability [1] to the cluster. This way, the overall likelihood will remain positive.

[1]In this paper, we simply assign $\frac{1}{|\Lambda|}$, where $|\Lambda|$ is the size of stream inputs observed so far.

```
Input: Stream with movement info. ($\Lambda = \{v_0 \dots v_n\}$)
Output: Stationary states $\mathbb{Z}$, Transitional states $\mathbb{S}$
Local: $z_{frm} \leftarrow NULL$, Duration $\gamma \leftarrow 0$
 1: $\mathbb{Z} \leftarrow \emptyset, \mathbb{S} \leftarrow \emptyset$
 2: for each $v_i$ in $\Lambda$ do
 3:   if $v_i.movement$ = moving then
 4:     $(\mathbb{S}, z_{frm}, \gamma) \leftarrow$ updateTrans($\mathbb{S}, z_{frm}, NULL, \gamma, t_i, move$)
 5:     continue next: $v_{i+1}$
 6:   $z \leftarrow \mathbb{Z}.classify(v_i)$
 7:   if $z \neq NULL$ then
 8:     $z.update\_parameters(v_i)$
 9:     $(\mathbb{S}, z_{frm}, \gamma) \leftarrow$ updateTrans($\mathbb{S}, z_{frm}, z, \gamma, t_i, stay$))
10:   else
11:     $\mathbb{Z}.newCluster(v_i)$
12: RETURN $(\mathbb{Z}, \mathbb{S})$
```

Figure 2: Incremental state learner

5.3 Modelling Transitional Behavior

While mining stationary states, we also incrementally learn and update a transitional state model, as shown in Figure 2, line 4 and line 9. The goal of this model is to predict future transitional states given past observations. To achieve this goal, we must first determine the temporal representation. In Section 5.3.1, we introduce a process for extracting time features. Based on these features, we propose an incremental learning algorithm in Section 5.3.2 that models the transitional behavior of a user.

5.3.1 *Temporal Feature Extraction*

A temporal feature has a fixed number of domain values that represent some semantic concepts of time. For example, a feature can have $\{morning, afternoon, evening\}$ as its domain values. Table 3 lists an example set of temporal features that are used in this paper. Notice that Each temporal feature has a unique ID, and each value of a feature also has a unique ID. In this paper, we use very general features, but domain specific features can be easily added.

At a given time instance t, it is possible to have multiple matching temporal feature-value pairs. In this case, we want to select only those that are discriminative or important. We use a precedence graph to encode the importance relationship. The graph defines a partial-order for all the temporal feature-value pairs. The extraction process then works as follows: map the time instance to all possible temporal feature-value pairs, then use the precedence graph to eliminate those values that are dominated.

The following example illustrates the process of extracting time features. Suppose the time instance "8 A.M. on Jan. 9th" is a normal workday morning. This instance matches two feature-value pairs: "F1:Morning" and "F4:Non-vacation". According to our precedence graph (Figure 3), "F1:Morning" is more discriminative than "F4:Non-vacation", so it is used as the the final time feature. Table 4 shows additional examples of this process.

In the end of the extraction process, only the set of important feature-value pairs $\{\beta_1 \dots \beta_k\}$ are left to represent the time instance t. In our experiments, we empirically evaluate different combinations of time features, and show that even with a few simple features, we can achieve very good prediction accuracy.

ID	Value ID:Value
F1	{v1:Morning, v2:Afternoon, v3:Night}
F2	{v4:Weekend, v5:Non-weekend}
F3	{v6:Weekend-morning, v7:Weekend-afternoon, v8: weekend-night}
F4	{v9:Vacation, v10:Non-vacation}

Table 3: Temporal features

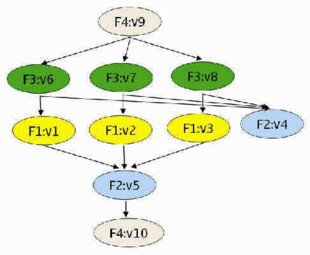


Figure 3: Temporal feature precedence graph

5.3.2 *Transitional State Learner*

There are two types of transitions: self-transition and transition between two different stationary states. In a transitional model, the state space $\mathbb{S}$ consists of all the possible pairs of stationary states (i.e., $\mathbb{S} = \{(z_i, z_j)\}$). Figure 4 outlines the algorithm for updating the transitional states. The algorithm takes inputs of the current transitional model ($\mathbb{S}$) and a pair of stationary states: the previous stationary state, z_{from}, and the current stationary state, $z_{current}$. The algorithm starts by evaluating the significance of the current stationary state $z_{current}$. The probability $Pr(z_{current})$ is also known as *support* [2]. If the support of the stationary state is bigger than a predefined minimum support, it is regarded as significant. (We evaluate different minimum support settings in Section 7.) If the stationary state is significant, and if the subject is moving at the point of recording, we do not know the destination yet. We cache the information of the from-state z_{from} and time information at line 5. Once the subject reaches a stationary state, if the movement cache z_{from} is not empty, we form a moving transition at line 8 (i.e., involving two stationary states). Otherwise, we update the self-transition (line 11) since we know it must transit from the same state. Given the time and transitional information, to reliably update the parameters $Pr(s_{i+1}|s_i)$ at line 12, we use the temporal features extractor $\mathcal{F}$ and past observations:

$$Pr(s_{i+1}|s_i) = Pr(s|s_i, \gamma, \mathcal{F}(t_i)) = Pr(s|s_i, \gamma, \beta_1 \dots \beta_k).$$

In this equation, γ represents the time duration of remaining in transitional state s_i. It is updated at line 3. A set of k temporal feature-value pairs $\{\beta_1 \dots \beta_k\}$ is obtained as illustrated in the previous section. Since the temporal feature-value pairs cannot dominate each other, they can be assumed to be conditional independent. We thus simplify the computation as:

$$Pr(s|s_i, \gamma, \beta_1 \dots \beta_k) = \frac{\prod_{j=1}^{k} Pr(s|s_i,\gamma,\beta_j)}{\sum_{q=1}^{m} \prod_{j=1}^{k} Pr(s_q|s_i,\gamma,\beta_j)}\ ^{2}$$

We now need to know the distribution of $Pr(s|s_i, \gamma, \beta_j)$. We first estimate the probability $Pr(s = s_i|s_i, \beta_j, \gamma)$, also known as ***survival probability***, which is the probability the subject stays in the same transitional state s_i at next time tick. Given a duration value from γ, we use a normal distribution to model it: $Pr(s = s_i|s_i, \gamma, \beta_j) = 1 - \int_{-\infty}^{\gamma} \mathcal{N}(\mu, \sigma)dx$ [3]. The parameter μ and σ can be learned from historical data, as illustrated in the following example. Assume the subject is currently in a stationary state z at time t_i. From the past data, for a time feature value of β_j (e.g., "F1:Morning"), we know that the subject stay in z for 20 minutes on average (i.e., $\mu = 20$) with standard deviation of 5 minutes (i.e., $\sigma = 5$). If the subject has already stayed in z for 25 minutes, the probability of remaining in z reduces to 16%, which also means that the probability of transiting to other states is 84%.

[2]To further simplify the computation, we assume the uniform conditional prior $Pr(\beta|s_i, \gamma)$ for the temporal features.

Time	Matched Value	Final Value
8AM Jan-09-09	{F1:v1, F4:v9}	{F1:v1}
8AM Jan-10-09	{F1:v1, F3:v5, F4:v9}	{F3:v6}
8AM Jan-12-09	{F1:v1, F4:v9}	{F4:v9}

Table 4: Example: Temporal feature extraction

```
Input: $\mathbb{S}$, $z_{form}$, $z_{current}$, $\gamma$, time tick: $t$, $movement$
Output: Transitional States $\mathbb{S}$, $z_{from}$, Duration $\gamma$
1:  $\gamma \leftarrow \gamma + t$
2:  if $z_{current} = NULL$ OR $Pr(z_{current}) < minsup$ then
3:      RETURN ($\mathbb{S}, z_{from}, \gamma$)
4:  if $movement$ = move then
5:      $z_{from} \leftarrow z_{from}$
6:  else
7:      if $z_{from} \neq NULL$ then
8:          $s \leftarrow (z_{from}, z_{current})$
9:          $z_{from} \leftarrow NULL$
10:     else
11:         $s \leftarrow (z_{current}, z_{current})$
12:     $\mathbb{S}.update\_parameters(s, \gamma, t)$
13:     $\gamma \leftarrow 0$
14: RETURN ($\mathbb{S}, z_{from}, \gamma$)
```

Figure 4: Algorithm UpdateTrans

Therefore, using the survival probability, we also know the total probability of going to other transitional states $Pr(s \neq s_i|s_i, \gamma, \beta_j)$. Depending on the meaning of s_i, the probability of transiting to other transitional states will be computed differently. In the first situation, s_i represents moving from one stationary state $z1$ to another stationary state $z2$. Since the destination is specified, $Pr(s \neq s_i|s_i, \gamma, \beta_j)$ represents the probability of reaching $z2$ (i.e., $s = (z2, z2)$). In the second situation, if s_i represents staying in a stationary state, $Pr(s \neq s_i|s_i, \gamma, \beta_j)$ represents the total probability mass of transiting to other transitional states: $\sum_{k=1 \wedge (k \neq i)}^{m} Pr(s = s_k|s_i, \gamma, \beta_j)$. If we know s_i is about to transit to s_k, the duration at s_i becomes irrelevant, and hence $Pr(s = s_k|s_i, \gamma, \beta_j)$ equals to $Pr(s = s_k|s_i, \beta_j)$, which can be modeled by a multinomial distribution. The parameters of the distribution can be maintained on-line when new observations arrived.

6. PREDICTION COMPONENT

Given a time window that consists of x number of time ticks. At a time tick t, the prediction task can be formalized as $Pr(s_{t+1} \dots s_{t+x}|s_0 \dots s_t)$, which represents the past state observations $s_0 \dots s_t$ and future observations $s_{t+1} \dots s_x$. Directly estimating this probability distribution is not feasible since the space for storing the joint distribution is large. In this paper, we make the first-order Markov chain assumption, such that the probability of observing state s at time tick t is only conditional dependent on the state at $t-1$.

[3]When the number of the observations for s_i is small, we use uniform distribution instead.

The prediction task can then be simplified as:
$Pr(s_{t+1} \ldots s_{t+x} | s_0 \ldots s_t) = \prod_{i=t}^{t+x} Pr(s_{i+1}|s_i)$
The first-order Markov assumption is reasonable since, in practice, our next location is highly associated with the current location, but less so for the previous locations. Since in Section 5.3.2, we have already derived an efficient algorithm for computing $Pr(s_{i+1}|s_i)$, the prediction task becomes feasible. We separate the prediction task into two sub-problems. The first problem is to predict the most probable transitional state s at current time t. The second problem is to predict the most probable state sequence for the time window x. If we can solve the first problem, we can use its solution for the second problem.

For the first problem, there are two possible scenarios. In the first scenario, at the point of prediction, the movement detector does not detect any movements, so the stationary learner will predict one of the stationary states z_i. Therefore, the transitional state s_t represents a self-transition $s_t = (z_i, z_i)$. In the second scenario, the movement detector reports the subject is on the move; at query time, we will not know s_t, we only know the last self-transitional state. In this case, the prediction starts from the last self-transitional state. The time interval from the last stationary state is discretized into a set of time ticks (e.g., every 10 minutes). Since we follow the first-order Markov assumption, we can apply the well-known Viterbi algorithm [10] to compute the most probable transitional state at query time t. The Viterbi algorithm is a dynamic programming algorithm for computing Viterbi path, which corresponds to a state sequence with maximum probability. Let us assume at query time t there are m time ticks since the last stationary state, and the best state at time t has probability: $P = \max(Pr(s_m))$. We can then define the states at each time tick i as: $Pr(s_i) = \max(Pr(s_i|s_{i-1}))$. For the starting state $s1$, we set $Pr(s_1 = z_1) = 1$ and $Pr(s_1 \neq z_1) = 0$ since it starts from a known stationary state z_1.

For the second problem, instead of returning the most probable transitional state, we return the Viterbi path. Given the query end time t, we divide the time from the last known stationary state into m time ticks, and output the state sequence with the maximum probability. In Figure 5, we illustrate a temporal range query. In this example, we have four time ticks. Each node in this graph represents a transitional state s. If s is a self-transitional state, we represent its stationary state only as z; otherwise, s is a pair of stationary states. In each node, we also keep the previous best node/state and the probability of reaching the current node. Therefore, at the end of the query time tick, we have the transitional state ($z3 \rightarrow z2$). If we backtrack using the previous best node, we can obtain the best sequence to be $\{(z1), (z1 \rightarrow z3), (z3), (z3 \rightarrow z2)\}$, which means that the subject will transit to the stationary state $z3$, and then start transiting to the stationary state $z2$.

7. EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate the effectiveness and efficiency of the proposed algorithms. In Section 7.1, we focus on the data collection and evaluation techniques. The prototype of our system has been implemented using Python. The run time reported in this paper are measured from a P4-2 GHz PC with Linux OS. For comparisons, the LDA and HMM models are generated utilizing the various statistical tools from Matlab.

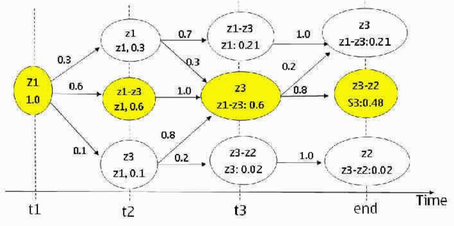


Figure 5: An example of Temporal Range Query

Type	User1		User2		User3	
	#rec	#dim	#rec	#dim	#rec	#dim
GSM	88,204	643	50,995	1,137	67,777	625
WiFi	41,995	2,936	23,818	3,669	34,921	3,718

Table 5: Data sets

7.1 Experimental Setup

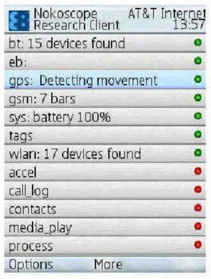
In this paper, we focus on GSM and WiFi sensor data, because these two types of data have clear location sense and are widely available. To evaluate the effectiveness of our approach, we have been continuously monitoring 3 users for a period of 66 days (12/25/2008 to 02/28/2009).

The data collection client is implemented using Python for S60, and runs on Nokia S60 series mobile phones. The client periodically uploads the data to a server. The overall service is also known as Nokia Simple Context service [1]. Figure 6 shows a screen capture of the client running on a Nokia N95 mobile phone. The client is highly customizable, with the capabilities of monitoring multiple sensors (GSM, WiFi, GPS, Bluetooth, and etc.) at the same time. One interesting feature of the client is an adaptive GPS, which uses the same movement detection algorithm described in this paper to control the GPS sensing. It can smartly turn on the GPS when the device is on the move, and switch it off otherwise. Although we only use the GPS data to geo-locate the WiFi and GSM signal sources, it is possible to further utilize the GPS data in the state detection algorithms, but we leave this for future work.

Figure 7 shows the server side interface of the last a few records of a user. Once the data is stored on the server, various APIs can be used to retrieve the data. Note that, although our models are built on a PC, the code can also be directly run on the mobile client in a real-time mode; however, we use a standard PC to benchmark the performance against other alternatives.

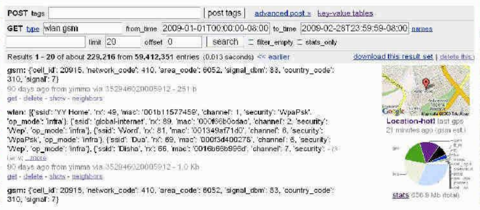
During the measurement time period, thousands of distinct WiFi access points and GSM cell IDs have been observed. Based on one user's data, if we geo-locate some of the GSM and WiFi signal sources (i.e., Figure 8 and Figure 9), we notice that they cover a very large spatial region. We also notice that it is not sufficient to use only individual signal sources (e.g., a GSM cell ID) to represent a stationary state, because many of these signal sources do not correspond to the stationary states (e.g., discovered during commutes), and many of them represents the same stationary states (e.g., multiple WiFi access points in one scan).

Table 5 summarizes the data sets recorded over a period of two months. The table shows the total number of records and the number of distinct signal sources for each user. Notice that the number of records for WiFi is less than GSM, this is because a WiFi scan is more power consuming than a GSM scan, and hence WiFi scans are done at slower rate than GSM scans. For our users, WiFi scanning rate is set at



```
Nokoscope Research Client        AT&T Internet 13:57
bt: 15 devices found
eb:
gps: Detecting movement
gsm: 7 bars
sys: battery 100%
tags
wlan: 17 devices found
accel
call_log
contacts
media_play
process
Options    More
```

Figure 6: Client



```
POST tags [          ] | post tags |  advanced post »  key-value tables
GET type [wlan gsm]  from_time [2009-01-01T00:00:00-08:00]  to_time [2009-02-28T23:59:59-08:00]  names
[          ]  limit [20]  offset [0]  | search |  filter_empty  stats_only
Results 1 - 20 of about 229,216 from 59,412,351 entries (0.013 seconds) << earlier    download this result set | delete this
gsm: {'cell_id': 20915, 'network_code': 410, 'area_code': 6052, 'signal_dbm': 83, 'country_code':
310, 'signal': 7}
90 days ago from yimma via 352946020005912 - 251 b
get - delete - show - neighbors
wlan: [{'ssid': 'YY Home', 'rx': 49, 'mac': '001b11577459', 'channel': 1, 'security': 'WpaPsk',
'op_mode': 'infra'}, {'ssid': 'global-internet', 'rx': 89, 'mac': '000f66b0cdac', 'channel': 2, 'security':
'Wep', 'op_mode': 'infra'}, {'ssid': 'Word', 'rx': 81, 'mac': '001349af71d0', 'channel': 6, 'security':
'WpaPsk', 'op_mode': 'infra'}, {'ssid': 'Dua', 'rx': 69, 'mac': '000f3d400278', 'channel': 6, 'security':
'Wep', 'op_mode': 'infra'}, {'ssid': 'Disha', 'rx': 66, 'mac': '0016b66b996d', 'channel': 7, 'security':
...more
90 days ago from yimma via 352946020005912 - 1.0 Kb
get - delete - show - neighbors
gsm: {'cell_id': 20915, 'network_code': 410, 'area_code': 6052, 'signal_dbm': 83, 'country_code':
310, 'signal': 7}
Location-hot! last gps
21 minutes ago (gsm est.)
stats 556.9 Mb (total)
```

Figure 7: Server

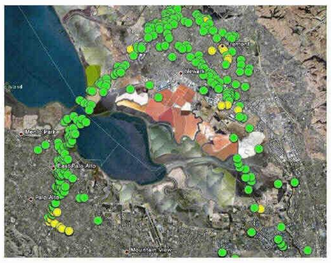


Figure 8: Cells

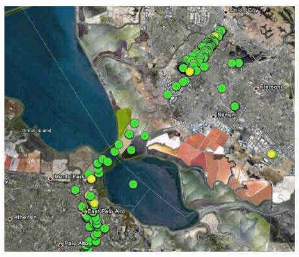


Figure 9: SSIDs

Component	Parameter	*Values*
Movement detection	movement threshold	10%
Stationary state learner	New cluster threshold	10%
Transitional state learner	Cluster Min Support	1%
	Temporal Features	Table 3
Prediction Module	Time tick	10 mins.

Table 6: Default parameters

2 to 5 minutes, and GSM scanning rate is set at 30 seconds to 2 minutes. In the reminder of this section, we focus on *User1*'s data to understand how well each component works. We use all users' data to summarize the overall performance.

In table 6, we show the default parameter settings used by *iLoc*. For different components, the parameters have different sensitivity. For instance, the threshold settings in movement detection and stationary state learner are less sensitive than the parameters used by transitional state learner. Therefore, in the later part of section, we evaluate performance sensitivity to the parameters.

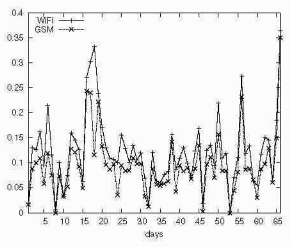


Figure 10: Movement ratio (days)

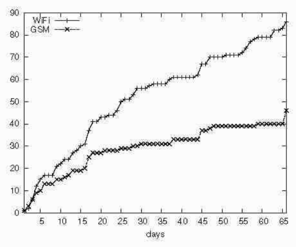


Figure 11: Num. stationary states (days)

7.2 Performance of Movement Detector

To show the effectiveness of the movement detector, in Figure 10, we show the daily movement ratio for *User1*. We can see that movements are not limited only to the weekends, but week days as well. After consulting the user, we confirm the movement detector can correctly capture large movements especially for commuting to different locations. In Table 7, we compute the average movement ratio per day for all users. We notice that the WiFi input stream can capture more movements than the GSM stream. This is because that a WiFi access point normally has smaller spatial coverage than a GSM cell, and hence more sensitive to the movements. However, the difference is not very large, which indicates GSM cells can also be used for movement detection.

7.3 Performance of Stationary State Learner

In Figure 11, we show the total number of the stationary states discovered by *User1* in different days. It clearly demonstrates that our stationary state learner can incrementally discover new stationary states as data arrives. We also notice that the number of WiFi states are larger than GSM states. This is also expected since many places nowadays have free WiFi access, but many of these access points might be under the same GSM cell. In Table 8, we summarize the total number of the stationary states discovered by all 3 users.

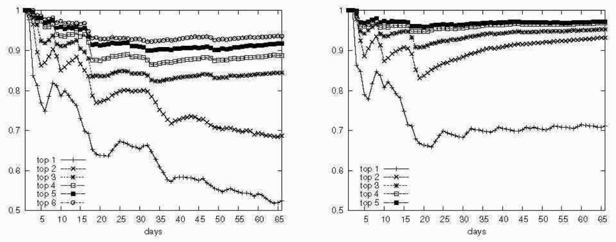


Figure 12: CDF of top states (GSM and WiFi)

The next question is that how these stationary states distributed. In Figure 12, we shows the cumulative probability of top-1 to top-k stationary states for *User1* in different days. Both GSM and WiFi has a big stationary state that consistently has more than 50% of the probability mass. After going through the state manually, we can clearly conclude it corresponds to the *Home* state. When we take top-5 for WiFi and top-6 for GSM, the cumulative probability is consistently more than 95% for each day. These states correspond clearly to *Work*, and other frequently visited places.

Type	*User1*	*User2*	*User3*
GSM	9.2%	16.2%	10.1%
WiFi	12.4%	24.9%	14.2%

Table 7: Avg. movement ratio per day

Type	*User1*	*User2*	*User3*
GSM	46	95	58
WiFi	86	109	83

Table 8: Total num. stationary states

Type	*User1*	*User2*	*User3*
GSM	6	23	10
WiFi	5	24	12

Table 9: Num. clusters est. for HMM and LDA

Since HMM and LDA learners are not incremental learners, to compare with them, we give the full advantage to

PTX-904-0007

GOOG-WSOU585-PA-00000482

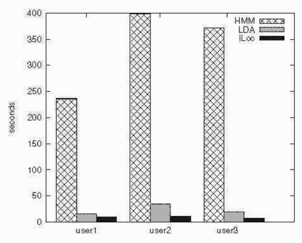


Figure 13: Model build time (GSM)

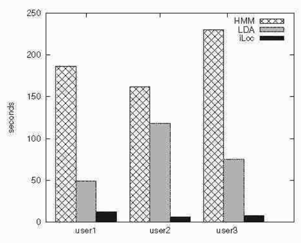


Figure 14: Model build time (WiFi)

these learners, such that we train HMM and LDA with the full set of data. Another difficult question is to decide the number of clusters for HMM and LDA. There are different heuristics proposed in the literature. In this paper, we can provide very good estimations by utilizing the results from *iLoc*. The number of clusters for HMM and LDA is based on the top stationary states that cover 95% of the stationary time of a user. The intuition is that, if HMM and LDA can outperform *iLoc*, it should discover a similar set of stationary states. We go through the process for each users, in Table 9, we show the number of clusters for different users. Lastly, to apply HMM and LDA effectively, we also need to determine the number of iterations. For HMM, we set the maximum number of iterations to be 100. For LDA, we use Gibbs sampler approach to training the LDA model [5]. Gibbs sampling approach has been shown to be more efficient than variational EM [3] and producing comparable results. The number of the iterations for the Gibbs sampler is set to be 500.

Figure 13 and Figure 14 show the model building time for each user. Even with 100 iterations, HMM takes the longest time to train. We also show that our incremental leaner is about 2 to 10 times faster than LDA with Gibbs sampling. The performance gain is mainly due to *iLoc* is an one-pass algorithm, and others require multiple iterations.

Next, we want to examine the quality of clusters from different learners. Since for *User1* we have decided to use top-5 clusters (WiFi stream) for HMM and LDA, in Table 10, we show the top-3 WiFi access points from each cluster algorithm. To the user, these access points can be easily associated with some semantic meaning. In Table 11, we ask the user to look at the top-10 access points and tag these cluster. For *iLoc*, the user is able to identify several interesting places, including home, work and a friend's home. Also, *iLoc* correctly clusters the WiFi access points near his usual lunch and Gym locations. Table 11 also shows the prior probabilities of each cluster, which represents the degree of importance of each cluster. After consulting with the user, he agrees with the assignment. However, HMM and LDA cannot clearly separate these states. For example, *Home* appears in a few clusters. If use these clusters as stationary states, it will severely affect the transitional model, and lead to poor prediction performance as we will show in the next section. We also go through the same exercise for other users, similar clustering behaviors are observed.

7.4 Performance of Transitional State learner

To fully test out the transitional model and the prediction framework, we form a continuous set of online prediction queries. In this online setting, we sequentially re-play the inputs from a data stream. At each sampling point, we apply

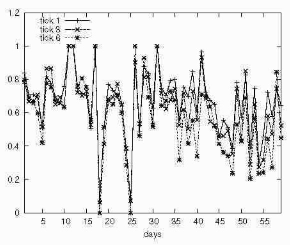


Figure 15: Accuracy Qt1 by days for *User1* (GSM)

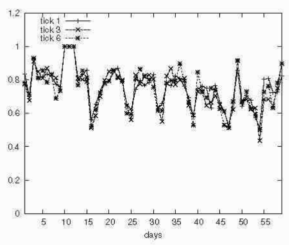


Figure 16: Accuracy Qt1 by days for *User1* (WiFi)

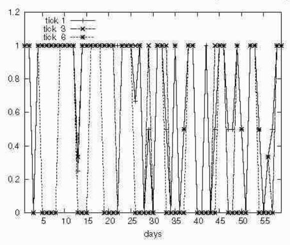


Figure 17: Accuracy Qt2 by days for *User1* (GSM)

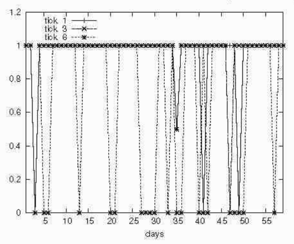


Figure 18: Accuracy Qt2 by days for *User1* (WiFi)

the movement detection algorithm, and two types of queries are formed.

Type	*User1*		*User2*		*User3*	
	Qt1	Qt2	Qt1	Qt2	Qt1	Qt2
GSM	7,973	272	4,832	142	5,292	102
WiFi	7,058	359	4,409	181	5,307	124

Table 12: Query load

The first query type is called **non-moving query** (i.e, *Qt1*), such that if a user is not moving, we form a time range query of 60 minutes. Our predictor, will predict at every 10 minutes intervals until the user starts to move again or the 60 minutes prediction window elapses. The accuracy will be computed at each time tick by comparing the predicted transitional state s against the stationary state z_0, such that it is correct if s represents the self-transition $s = (z_0, z_0)$.

The second query type is called **moving query** (i.e., *Qt2*), such that if the user is on the move, starting from the most recent stationary state z_0, we form a time range until the user reaches another stationary state z_1. During this time range, we predict at every 10 minutes interval. The accuracy will be measured at each time tick by comparing the predicted transitional state s against the correct transition between two stationary states $z_0 \rightarrow z_1$ (i.e., if $s = (z_0, z_1)$). Table 12 shows the query load for both types of queries.

In fact, these two query types cover many use cases. A non-moving query essentially asks how long a user will be staying in the same stationary state, and a moving query asks if a user is about to transit and to which stationary state. If *iLoc* can consistently provide accurate predication for both queries, it can be utilized for many real-life applications (e.g., Ad-serving).

Although our models are fully incremental models, we do not apply the prediction tests on the initial days (i.e., from 12/25/2008 to 12/31/2008). We only use these a few days to bootstrap the models. This way, we start with a reasonable prediction model. The predictions are performed continuously on exactly two months of data (i.e., from 01/01/2009 to 2/28/2009).

For the predictions, if a predictor refuses to make a prediction, we consider the predictor does not have sufficient

Methods	*State1*	*State2*	*State3*	*State4*	*State5*
iLoc	YY Home, Disha, Dua	WANO, Kelevala, noklabwl	HotelWireless, Multi-Point_Acess, SEASONS_INN	Aquarium, linksys, Arthur	glenbrook, homelinksys, Kwan AP
HMM	YY Home, Disha, HotelWireless	Kelevala, WANO, noklabwl	YY Home, HotelWireless, Dua	Aquarium, dungeon, Due Net	Aquarium, dungeon, WANO
LDA	WANO, Kelevala, noklabwl	YY Home, Disha, Gammavista	Disha, YY Home, Dua	Lee9, YY Home, Gamavista	YY Home, Dua, Disha

Table 10: Top 3 SSID (WiFi)

Methods	*State1*	*State2*	*State3*	*State4*	*State5*
iLoc	Home (71%)	Work (22%)	Lunch (2%)	Friend Home (1.5%)	Gym (0.3%)
HMM	Home (49%)	Work (26%)	Home (18%)	Friend Home, Work (6%)	Friend Home, Work (1%)
LDA	Work (24%)	Home (19%)	Home (19%)	Home (19%)	Home (19%)

Table 11: State Location (WiFi)

	State1	*State2*	*State3*	*State4*	*State5*
State1	37%	0%	63%	0%	0%
State2	0%	99%	0.9%	0.1%	0%
State3	82%	10%	1%	7%	0%
State4	0%	0.5%	0.5%	36%	63%
State5	0%	0%	0%	56%	44%

Table 13: HMM State Transition Matrix (WiFi)

information to make the prediction, and it is not part of the query load. This is reasonable since the goal is to have very accurate predictions rather than very broad ones. To gain some insights to the prediction process, we first take a detailed look at daily prediction for *User1*. If there's no predictions made for a day, we treat it as correct choice (i.e., 100% correct for the day).

In Figure 15 and Figure 16, we show the average prediction accuracy on each day for *User1* with the query type *Qt1*. We noticed that predictions based on GSM stream is not as good as WiFi stream. The main reason is that GSM cells cover larger spatial regions. For *Qt1*, it mainly tests the self-transitions for different stationary states. The stationary states from GSM stream can sometimes mix-up real stationary states if they are nearby. As the result, the self-transition model for GSM is not as good as the one from WiFi. However, even in this situation, for GSM, *iLoc* is still able to achieve 62% overall accuracy. For WiFi stream, it performs very well, and achieves 73% overall accuracy.

Similarly, in Figure 17 and Figure 18, we show the average prediction accuracy on each day for *User1* for query type *Qt2*. We observe that for some days, the accuracy is not so good (i.e., close to 0). After consulting with the users, it turns out that these days mostly are weekends and vacations. Predicting these days are challenging due to irregular movements. However, being able to differentiate these days from normal week days can improve the prediction accuracy. Another observation is that GSM predictor is again worse than WiFi predictor. Therefore, it is important that the stationary learner can discover the stationary states accurately. Overall, *iLoc* predictor based on WiFi stream is able to achieve 84% prediction accuracy, and GSM stream gets 25%. To clearly show the impact of the query time window, we only plot the query time windows at 10, 30, and 60 minutes (i.e., tick1, tick3, and tick6). Since the lines are largely overlapped for both types of queries, it means both predictors (GSM and WiFi) can make reasonable predictions for the time range up to an hour. This is sufficient to many application scenarios, for which predicting a user's behavior for the next hour is critical for the recommendation purposes.

After having some understandings on how prediction works for one user, in Table 14, we show the overall accuracy measures for all users under different approaches, and different parameter settings. The first row shows the accuracies of *iLoc* with the default parameters (i.e., Table 6). If a method has better performance for a query setting, the result is highlighted. This way, we can easily compare the performance of different approaches.

Table 14 has three parts. In the first part, we compare the performance of *iLoc* against HMM and LDA. Both HMM and LDA are not suitable for our prediction task. LDA only provides clustering information, which can be used as stationary states. HMM requires observations to infer the hidden states, whereas for us, at the query time instance, we do not have future observations yet, but we would like to infer the state sequence. Although HMM provides transitional probability distribution for different states, these transitions are mostly for self-transitions. If we would run Viterbi algorithm, it will always end up with the same state. To illustrate this, in Table 13, we show the transition matrix from a HMM model. From the matrix itself, we can observe some transitions other than self-transition. For instance, *state1* is able to transit to *state3* with 63% of probability. However, given the semantic meaning of the states shown in Table 11, we notice that they are essentially the same state. Similar conclusions can be drawn for other transitions. Therefore, to have a fair comparison, we only use the cluster/stationary states from LDA and HMM, and plug them into our transitional model and prediction framework. As we can observe from the results, especially for the query type *Qt2*, instead of unable to predict transitional behavior, we have some predictions for HMM and LDA. However, the accuracy is still far behind the *iLoc* with default parameters (i.e., *iLoc*-default in the table). This can also be observed from the last column of the table, which shows the average accuracy for the corresponding method. We would like to note that since the full set of data are used to train HMM and LDA, any incremental learning methods for HMM and LDA cannot outperform the results shown in this paper. Furthermore, being able to easily plug-in different clustering methods, we have demonstrated the flexibility and openness of our overall framework.

The second part of the Table 14 shows the accuracy of *iLoc* when we vary the min-support parameter. We have tested 4 settings (0%, 1% default, 2% and 5%), which correspond to *iLoc*-0, *iLoc*-default, *iLoc*-2, and *iLoc*-5. The higher the min-support, less number of stationary states are considered

	User1				*User2*				*User3*				
Methods	Qt1 *GSM*	Qt1 *WiFi*	Qt2 *GSM*	Qt2 *WiFi*	Qt1 *GSM*	Qt1 *WiFi*	Qt2 *GSM*	Qt2 *WiFi*	Qt1 *GSM*	Qt1 *WiFi*	Qt2 *GSM*	Qt2 *WiFi*	Average
iLoc-default	**62%**	**73%**	**25%**	**84%**	**62%**	**64%**	**26%**	**44%**	**71%**	**79%**	**68%**	**81%**	**61%**
LDA	49%	72%	18%	22%	55%	**71%**	6%	2%	61%	68%	5%	11%	37%
HMM	42%	66%	**42%**	21%	46%	64%	20%	8%	58%	73%	2%	21%	39%
iLoc-0	59%	65%	9%	42%	61%	**67%**	6%	25%	70%	74%	42%	41%	47%
iLoc-2	**68%**	72%	**27%**	**90%**	61%	63%	**35%**	**50%**	73%	79%	**94%**	**91%**	**67%**
iLoc-5	59%	73%	**58%**	**95%**	48%	60%	**52%**	**72%**	68%	79%	**81%**	**93%**	**70%**
iLoc-none	**68%**	**76%**	10%	16%	66%	66%	13%	35%	83%	84%	8%	10%	45%
iLoc-man	62%	72%	25%	**86%**	60%	63%	24%	40%	66%	69%	66%	81%	60%

Table 14: Overall accuracy for different methods

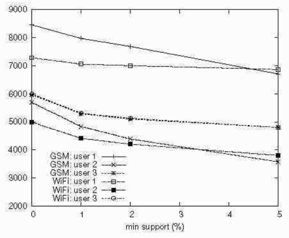


Figure 19: Num. Qt1

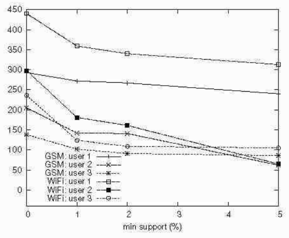


Figure 20: Num. Qt2

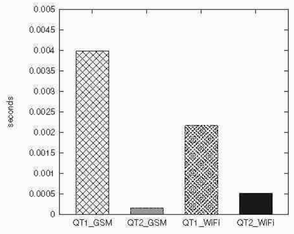


Figure 21: Query Ex. Time

in the transitional model, and generally increases the prediction accuracy with the price of not being able to predict some infrequent movements. To further study the tradeoffs, in Figure 19 and Figure 20, we show the number of queries for different users with different data streams. We notice that as min-support increases, the number of the queries that can be answered monotonically decreases. At min-support 0%, *iLoc* can answer maximal amount of queries both for *Qt1* and *Qt2*, but with the worst accuracy. For *Qt1*, with min-support at 1%, *iLoc* is still able to answer a large portion of queries with good gain in accuracy. For *Qt2*, at 1% min-support, the number of queries dropped can be large for some settings. However, the accuracy gain is dramatic. As we keep increasing the min-support (i.e., *iLoc*-2, and *iLoc*-5), the number of queries dropped also increases, but the accuracy gain is only moderate (6% to 9% overall improvements). Therefore, to reach the right balance, we choose to use 1% min-support as our default.

The third part of the Table 14 shows the effect of the temporal features. The *iLoc*-none uses no temporal features, it performs badly for *Qt2* queries. This is expected since it does not differentiate movements from time dimension. When we add the morning, afternoon and night feature (i.e., iLoc-man), the performance improves drastically, and further enhanced with the full feature set (i.e, *iLoc*-default).

Lastly, we would like to demonstrate the efficiency of the prediction algorithm. In Figure 21, we show the average speed for executing a prediction query. Most of the queries complete within 5 millisecond. With this speed, we are very confident that *iLoc* can scale nicely as a web service.

8. CONCLUSION AND FUTURE WORK

In this paper we presented our approach for learning and refining the location state models. The *iLoc* framework is not only able to produce up-to-date location models, but is also able to predict future location states. We have conducted an extensive set of experiments to demonstrate the effectiveness and the efficiency of our solution.

There are several future research directions that we would like to pursue. Firstly, we would like to incorporate more sensor inputs to our framework. For instance, we are considering to use accelerometer sensor inputs to differentiate finer grain movements. Secondly, we would like to incorporate signal strength to help us identify smaller stationary states. Last but not least, we would like to conduct larger trials to test out our framework.

9. REFERENCES

[1] Nokia simple context service http://simplecontext.com/eb2. In *Nokia Research Center, Palo Alto.*

[2] R. Agrawal and R. Srikant. Fast algorithms for mining assocation rules. In *VLDB Conference*, 1994.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 2003.

[4] M. Chen, T. Sohn, D. Chmelev, D. Haehnel, J. Hightower, J. Hughes, A. LaMarca, F. Potter, I. Smith, and A. Varshavsky. Pracical metropolitan-scale positioning for gsm phones. In *UbiComp*, 2006.

[5] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of National Academy of Sciences*, 2004.

[6] J. Hightower, S. Consolvo, A. LaMarca, I. Smith, and J. Hughes. Learning and recognizing the place we go. In *UbiComp*, 2005.

[7] F. Jelinek. Statistical methods for speech recognition. In *MIT Press*, 1997.

[8] M. Meila and D. Heckerman. An experimental comparison of model-based clustering methods. In *Machine Learning*, 2001.

[9] V. Otsason, A. Varshavsky, A. LaMarca, and E. Lara. Accurate gsm indoor localization. In *UbiComp*, 2005.

[10] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 1986.

[11] G. Salton and M. McGill. Introduction to modern information retrieval. In *McGraw-Hill*, 1983.

[12] A. Schwaighofer, M. Grigoras, V. Tresp, and C. Hoffmann. Gpps: A gaussian process positioning system for cellular networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[13] D. D. Walker and E. K. Ringger. Model-based document clustering with a collapsed gibbs sampler. *ACM SIGKDD Conference*, 2008.